OCR Computer Science A Level

2.2.2 Computational Methods

Advanced Notes

Specification:

2.2.2 a)

- **Features that make a problem solvable by computational methods**

2.2.2 b)

- **Problem recognition**

2.2.2 c)

- **Problem decomposition**

2.2.2 d)

- **Use of divide and conquer**

2.2.2 e)

- **Use of abstraction**

2.2.2 f)

- **Solving problems using:**
  - Backtracking
  - Data mining
  - Heuristics
  - Performance modelling
  - Pipelining
  - Visualisation

Features that make a problem solvable by computational methods

Not all programs can be solved using computers. The first stage of problem solving is identifying whether or not a problem can be solved using computational methods. A problem that can be solved using an algorithm is computable. Problems can only be called computable if they can be solved within a finite, realistic amount of time. Problems that can be solved computationally typically consist of inputs, outputs and calculations.

Although some problems are computable, it may be impractical to solve them due to the amount of resources or length of time they require in order to be completed. The number of problems that are in fact solved computationally are constrained, therefore, by factors such as processing power, speed and memory. Breakthroughs in technology have made it feasible to solve more problems computationally than ever before.

Problem recognition

Once a problem has been determined to be computable, the next stage is to clearly identify what the problem is. Stakeholders state what they require from the finished product and this information is used to clearly define the problem and the system requirements. Requirements may be defined by:

- Analysing strengths and weaknesses with the current way this problem is being solved
- Considering types of data involved including inputs, outputs, stored data and amount of data

Problem decomposition

Once a problem has been clearly defined, it is continually broken down into smaller problems. This continues until each subproblem can be represented as a self-contained subroutine. This technique is called problem decomposition and aims to reduce the complexity of the problem by splitting it up into smaller sections which are more easy to understand. By identifying subproblems, programmers may find that certain sections of the program can be implemented using pre-coded modules or libraries which saves time which would otherwise have been spent on coding and testing.

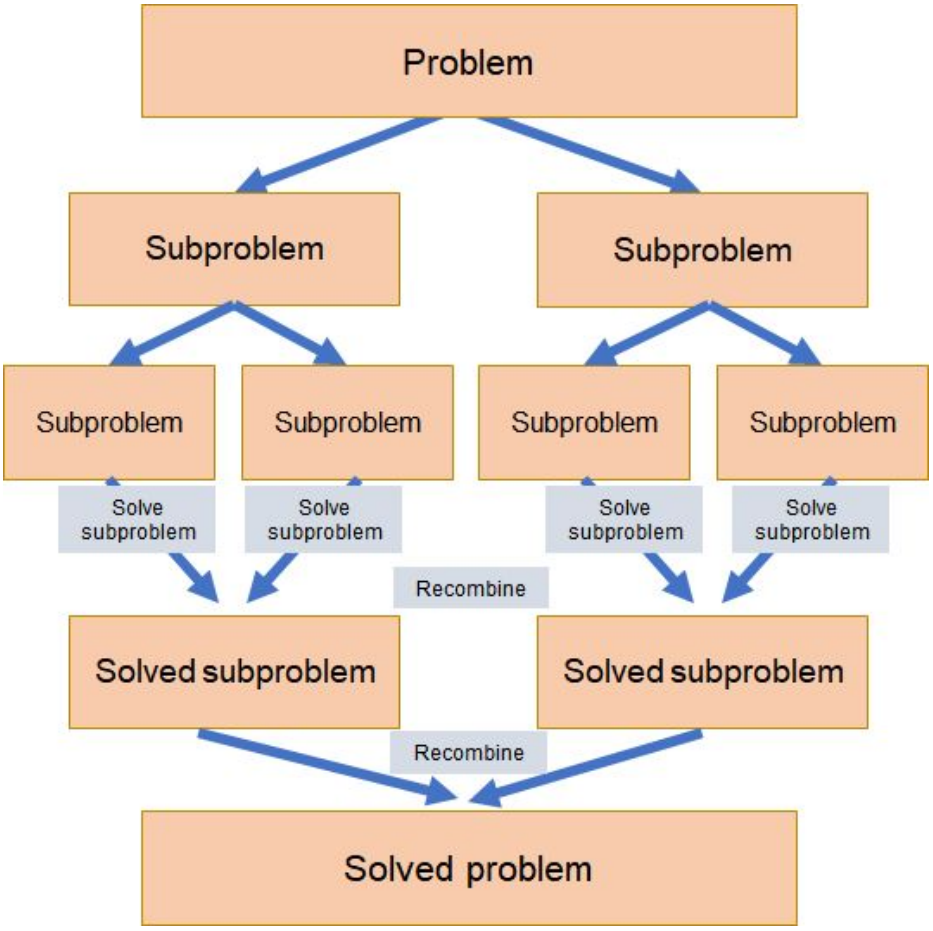
Synoptic Link

Top-down design discussed in **2.2.1** is an example of **problem decomposition**.

Decomposition also makes the project easier to manage, as different software development teams can be assigned separate sections of the code according to their specialisms. These can be individually designed, developed and tested before being combined to produce a working piece of software at the end. Decomposition enables multiple parts of the project to be developed in parallel, making it possible to deliver projects faster. This technique makes debugging simpler and less-time consuming, as it is easier to identify, locate and mitigate errors in individual modules. Without problem decomposition, testing can only be carried out once the entire application has been produced therefore making it hard to pinpoint errors.

Use of divide and conquer

Divide and conquer is a problem-solving technique used widely across computer science. This strategy can be broken down into three parts: divide, conquer and merge. 'Divide' involves halving the size of the problem with every iteration. Each individual subproblem is solved in the 'Conquer' stage, often recursively. The solutions to the subproblems are then recombined during the 'Merge' stage to form the final solution to the problem.



One common use of divide and conquer is in binary search, in which the middle value of the sorted list to be searched is compared to the value to be found. If this value is smaller than the search value, the lower half of the list is discarded and the upper half of the list is recursively searched using the same technique: the midpoint is found and half of the list is discarded. If the middle value is found to be larger than the search value, the upper half of the list is discarded and the lower half is recursively searched. This continues until either the search value is found or the list is broken down until it contains one element, which means that the value does not exist in the list.

Synoptic Link

Binary search and other **divide and conquer** algorithms are covered in **2.3.1 Search Algorithms**.

Divide and conquer is applied to problem-solving in quick sort and merge sort. The principle of divide and conquer is also used in problems which can be reduced by less than half in every iteration. This technique is sometimes called 'Decrease and Conquer'.

The biggest advantage of using divide and conquer to solve problems is that the size of the problem is halved with each iteration which greatly simplifies very complex problems. This means that as the size of a problem grows, the time taken to solve it will not grow as significantly. The number of recursive calls made by a function that halves with each iteration is $\log_2(n)$. Therefore, the time complexity of algorithms that use divide and conquer is of the order $O(\log n)$. As divide and conquer mostly makes use of recursion, it faces the same problems that all recursive functions face: stack overflow will cause the program to crash and large programs are very difficult to trace.

Synoptic Link

Time complexity and **sorting algorithms** are discussed further in **2.3.**

Use of abstraction

Representational abstraction is a powerful technique that is key to solving a problem computationally. This is when excessive details are removed to simplify a problem. Often, problems may be reduced down until they form problems that have already been solved which allows pre-programmed modules and libraries to be used rather than coding from scratch. Abstraction allows programmers to focus on the core aspects required of the solution rather than worrying about unnecessary details.

Synoptic Link

You will have already encountered **abstraction** in **2.1.1**.

Using levels of abstraction allows a large, complex project and its functionality to be split up into simpler component parts. Individual components can then be dealt with by different teams, with details about other layers being hidden. This technique makes projects more manageable.

Abstraction by generalisation may also be used to group together different sections of the problem with similar underlying functionality. This allows for segments to be coded together and reused, so saves time. Once a problem has been abstracted into levels, abstract thinking is required to represent real-world entities with computational elements, such as tables and variables.

Problem solving strategies

There are several other techniques that may be used to solve problems computationally:

Backtracking

Backtracking is a problem-solving technique implemented using algorithms, often recursively. It works by methodically visiting each path and building a solution based on the paths found to be correct. If a path is found to be invalid at any point, the algorithm backtracks to the previous stage and visits an alternate path. Depth-first graph traversals are an example of backtracking.

This process can be visualised using the idea of a maze. There are many possible routes through a maze but only few lead to the correct destination. A maze is solved by visiting each path and, if a path leads to a dead end, returning back to the most recent stage where there are a selection of paths to choose from.

Data mining

Data mining is a technique used to identify patterns or outliers in large sets of data, termed big data. Big data is typically collected from a variety of sources. Data mining is used in software designed to spot trends or identify correlations between data which are not immediately obvious.

Insights from data mining can be used to make predictions about the future based on previous trends. This makes data mining a useful tool in assisting business and marketing decisions. Data mining may uncover, for example, which products are bought at certain points of the year and this information can help shops prepare enough stock in advance.

Data mining has also been used to reveal insights about people's shopping habits and preferences based on their personal information. These insights can be used to inform marketing techniques.

However, as data mining often involves the handling of personal data, it is crucial that it is dealt with in accordance with the present legislation regarding data protection. As of 2018, all data held and processed by organisations within the EU must follow the rules set by the GDPR.

Synoptic Link

You will have learnt about laws surrounding **data protection** in **1.5.1**.

Heuristics

Heuristics are a non-optimal, 'rule-of-thumb' approach to problem-solving which are used to find an approximate solution to a problem when the standard solution is unreasonably time-consuming or resource-intensive to find. The solution found through using heuristics is not perfectly accurate or complete; however, the focus is on finding a quick solution that is 'good enough' and heuristics provide a shortcut to this.

Heuristics are used to provide an estimated solution for intractable problems such as the renowned Travelling Salesman Problem as well as the A* algorithm, and are also used in machine learning and language recognition.

Intractable problems

Problems for which the solutions **takes an unreasonably long time** to be found.

Performance modelling

Performance modelling eliminates the need for true performance testing by providing mathematical methods to test a variety of loads on different operating systems. Performance modelling provides a cheaper, less time-consuming or safer method of testing applications. This is useful for safety-critical computer systems, such as those used on an airplane, where it is not safe to do a real trial run before the system can be implemented. The results of performance modelling can help companies judge the capabilities of a system, how it will cope in different environments and assess whether it is safe to implement.

Pipelining

Pipelining is a process that allows for projects to be delivered faster, as modules are divided into individual tasks, with different tasks being developed in parallel. Traditionally, the output of one process in pipelining becomes the input of another, resembling a production line.

Synoptic Link

Pipelining is common in **RISC processors** in which the different **sections of the Fetch-Decode-Execute cycle** are performed **simultaneously.** You will have come across this **1.1.1**.

Visualisation

Data can be presented in a way that is easier for us to understand using visualisation. This makes it possible to identify trends that were not otherwise obvious, particularly amongst statistical data. Depending on the type of data, data may be represented as graphs, trees, charts and tables. Visualisation is another technique that is used by businesses to pick up on patterns which can be used to inform business decisions.